

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平10-143380

(43) 公開日 平成10年(1998) 5月29日

(51) Int.Cl.⁶

識別記号

F I

G 0 6 F 9/46
15/163 6 0
3 7 0G 0 6 F 9/46
15/163 6 0 B
3 7 0 N

審査請求 未請求 請求項の数 6 O L (全 10 頁)

(21) 出願番号

特願平8-295185

(22) 出願日

平成 8 年(1996)11月 7 日

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目 6 番地

(71) 出願人 000153443

株式会社日立情報制御システム

茨城県日立市大みか町 5 丁目 2 番 1 号

(72) 発明者 舟越 和己

茨城県日立市大みか町五丁目 2 番 1 号 株

式会社日立情報制御システム内

(72) 発明者 田村 滋

茨城県日立市大みか町五丁目 2 番 1 号 株

式会社日立製作所大みか工場内

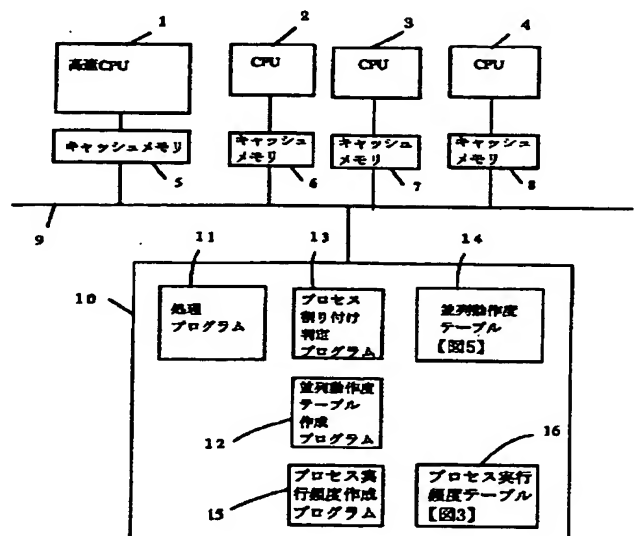
(74) 代理人 弁理士 高崎 芳紘

(54) 【発明の名称】 マルチプロセッサシステム

(57) 【要約】

【課題】 逐次型プログラムを高速CPUと低速CPUとに割り付けて、効率的な処理をはかりたい。

【解決手段】 1台の高速CPU1と複数台の低速CPU2、3、4と、共通バス9、共有メモリ10とより成るマルチプロセッサシステムより成る。高速CPU1と低速CPU2、3、4とに処理プログラム11のプロセスP₁～P_nを並列動作度の大小及び処理時間の大小に応じて割り付ける。



【特許請求の範囲】

【請求項 1】 逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持ち、プログラムを構成するプロセスの並列動作度及び処理時間の大小によりプロセスの各プロセッサへの配分を行うことを特徴とするマルチプロセッサシステム。

【請求項 2】 逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持ち、プログラムを構成するプロセスの並列動作度及び処理時間をオフラインで求めてメモリに登録しておき、プログラム実行時に上記プロセスの並列動作度及び処理時間の大小により、プロセスの各プロセッサへの配分を行うものとしたことを特徴とするマルチプロセッサシステム。

【請求項 3】 請求項 1 又は 2 において、プログラムを構成するプロセスの並列動作度及び処理時間は、すべてのプロセスの対について、2つのプロセッサで同時実行して求めた並列処理可能度及び実行頻度とから得るものとしたマルチプロセッサシステム。

【請求項 4】 逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持つマルチプロセッサシステムにおいて、逐次型言語で作成されたプログラムについて、そのプログラムを構成するプロセスの並列動作度及び処理時間の大小により、プロセスの各プロセッサへの配分の割り付けを行うマルチプロセッサシステム。

【請求項 5】 逐次処理用の高速プロセッサと並列処理用の複数のプロセッサとを持ち、逐次型言語で作成されたプログラムを実行するマルチプロセッサシステムにおいて、

プログラムを構成するプロセスについて求めた並列動作度及び処理時間をメモリに格納しておき、上記プログラムのプロセス実行待ちキューの中の先頭のプロセスについて、メモリに格納してあるプロセスの並列動作度から得られる並列度の大小を比較し、小であれば高速プロセッサが空くのを待って当該プロセスを高速プロセッサに割り付け、大であればプロセッサ数の並列度大のプロセスをキューより取り出しその各プロセッサについての上記メモリに格納してある処理時間を参照して実行させるべき並列度大のプロセスを処理時間順に並べ、全プロセッサが空くのを待って、処理時間が最長のプロセスを高速プロセッサに割り当て、残りを並列プロセッサに割り

当てるものとしたマルチプロセッサシステム。

【請求項 6】 高速プロセッサと、複数の低速プロセッサと、各プロセッサ対応のキャッシュメモリと、キャッシュメモリに接続された共通バスと、この共通バスに接続された共有メモリと、を有すると共に、共通メモリ上でプログラムを構成するプロセスの並列化の度合いを示すテーブルをオフライン的に作成し、このプロセスの並列度の度合い及び処理時間の大小によりプロセスをプロセッサに割り当て対応キャッシュメモリにその旨の連絡及び対応データを送出するものとしたマルチプロセッサ

システム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、電子計算機に係わり、特に複数のプロセッサを備え1つの演算処理を並列に高速処理するマルチプロセッサシステムに関する。

【0002】

【従来の技術】マルチプロセッサシステムは、逐次型言語（例えばFORTRANやC言語）を用いて作成されたプログラムに対し、自動並列化コンパイラ等により並列処理可能な処理を決め、プログラムを実行している。例えば1つのプログラム中に繰り返し処理であるDOループが存在する場合、このDOループを複数のCPUで処理している。例えば、処理プログラム中に

DO I = 1, 40

A(I) = B(I) + C(I)

なる演算がある場合、I = 1 ~ 10、I = 1 ~ 20、I = 21 ~ 30、I = 31 ~ 40の演算を各々別のCPU（例えば別の4つのCPU）に割り当て、並列処理している。

【0003】また、さらに進んだ方式では、最初から並列処理用言語を使用して複数プロセスの並列性記述によりプログラムを作成し、自動並列化コンパイラで対応困難なプロセス間の並列処理を実現している。並列処理用言語は、例えば、「並列処理技術、1991年、コロナ社発行」の第105頁から第113頁において論じられているように、並列実行可能プロセスの記述等をコンパイラだけに頼らずに積極的にプログラマの知見を利用し効率良い並列処理を実現しようとするアプローチである。前記で作成されたプロセスをマルチプロセッサへ割り付ける方法としては、従来のマルチプロセッサシステムでは、複数あるCPUの稼働率を高めるため、逐次処理を行うCPUや並列処理を行うCPUを常にダイナミックに変えている。

【0004】また、別の従来のマルチプロセッサシステムでは、ある1つのプログラムを実行中だけ、逐次処理ならばn個中の任意の1個のCPUに固定し、並列処理ならば並列処理を行うCPUを固定的に割り当て、逐次処理と並列処理が動作するタイミングをフラグによりとる（逐次処理実行中は並列処理は動作せず、並列処理が全て終了するまでは次の逐次処理は動作しないということフラグにより管理）ことにより、ハード量が少なくオーバーヘッドの小さいマルチプロセッサシステムを実現している。ここで、マルチプロセッサは、同じ性能のn個のプロセッサからなっている。この従来例としては、特開平2-144657号がある。

【0005】

【発明が解決しようとする課題】まず第1に並列可能な処理の決定について、上記従来技術の自動並列化コンパイラは、DOループの並列処理化には有効であるが、プ

ロセス間の並列処理化に対しては適していない。これに対しては、並列処理用言語によるプログラミングがあるが、この方法では従来の逐次型言語のプログラムをそのままでは使用できない。次に、プロセスをCPUへ配分する方式については、上記従来技術のように、逐次処理を行うCPUや並列処理を行うCPUを常にダイナミックに変えるように構成すると、CPUの稼働率は高くなるが、ダイナミックに変動させるためのハード量が増大してしまい、また各CPUの同期用のハード量も増大してしまうという問題が生じる。さらに、オーバーヘッドが大きくなるという不都合も生じる。

【0006】また、プログラムを実行する間だけそれを行うCPUを固定する方法は、オーバーヘッドは少ないが、実行するCPUの選択は空いているCPUが選ばれるのみなので、以下の理由により当該プログラムを実行するのに適切なCPU資源が選ばれるとは限らない。プログラムで対象とする通常の業務処理は、逐次処理と並列処理が混在している場合が一般的である。例えば、電力システムの長期計画業務（例：年間計画）では、期間内の潮流図を日付を変えて繰り返し作成する処理（並列可能な処理）と、この結果に対して系統解析を行う処理（逐次処理）とに分けられる。この場合、逐次処理は1台のCPUで実行されるので出来るだけ高速なCPUがよい。一方において、並列処理は、複数CPUで実行されるので高速でなくても台数効果が出せる。従って、従来の同じ性能のマルチプロセッサ構成よりもCPUの性能を逐次用の高速プロセッサと並列用の複数のプロセッサにしたマルチプロセッサ構成の方が有効である。

【0007】本発明の目的は、従来の逐次型言語のプログラムをそのまま使用してプロセス間の並列処理を容易に実現するマルチプロセッサシステムを提供するものである。さらに本発明の目的は、逐次処理と並列処理が混在するプログラムを逐次用の高速プロセッサと並列用の複数のプロセッサという2種類のハード資源に有効に配分可能にして、トータルとして効率的なプロセッサ管理や効率的なプログラム処理の実現をはかるマルチプロセッサシステムを提供することにある。

【0008】

【課題を解決するための手段】本発明は、逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持ち、プログラムを構成するプロセスの並列動作度及び処理時間の大小によりプロセスの各プロセッサへの配分を行うことを特徴とするマルチプロセッサシステムを開示する。

【0009】更に本発明は、逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持ち、プログラムを構成するプロセスの並列動作度及び処理時間をオフラインで求めてメモリに登録しておき、プログラム実行時に上記プロセスの並列動作度及び処理時間の大小により、プロセスの各プロセッサへの配分を行うものとした

ことを特徴とするマルチプロセッサシステムを開示する。

【0010】さらに本発明は、プログラムを構成するプロセスの並列動作度及び処理時間は、すべてのプロセスの対について、2つのプロセッサで同時実行して求めた並列処理可能度及び実行頻度とから得るものとしたマルチプロセッサシステムを開示する。

【0011】更に本発明は、逐次処理用の高速プロセッサと並列処理用の複数のプロセッサを持つマルチプロセッサシステムにおいて、逐次型言語で作成されたプログラムについて、そのプログラムを構成するプロセスの並列動作度及び処理時間の大小により、プロセスの各プロセッサへの配分の割り付けを行うマルチプロセッサシステムを開示する。

【0012】更に本発明は、逐次処理用の高速プロセッサと並列処理用の複数のプロセッサとを持ち、逐次型言語で作成されたプログラムを実行するマルチプロセッサシステムにおいて、プログラムを構成するプロセスについて求めた並列動作度及び処理時間をメモリに格納しておき、上記プログラムのプロセス実行待ちキューの中の先頭のプロセスについて、メモリに格納してあるプロセスの並列動作度から得られる並列度の大小を比較し、小であれば高速プロセッサが空くのを待って当該プロセスを高速プロセッサに割り付け、大であればプロセッサ数の並列度大のプロセスをキューより取り出しその各プロセッサについての上記メモリに格納してある処理時間を参照して実行させるべき並列度大のプロセスを処理時間順に並べ、全プロセッサが空くのを待って、処理時間が最長のプロセスを高速プロセッサに割り当て、残りを並列プロセッサに割り当てるものとしたマルチプロセッサシステムを開示する。

【0013】

【発明の実施の形態】以下、本発明の実施の形態を図面を参照して説明する。図1は、本発明の実施の形態に関するマルチプロセッサシステムの構成図である。本実施の形態のマルチプロセッサシステムは、逐次処理と並列処理を効率的に実行できるハード構成を目的として逐次処理用の高速プロセッサ（CPU）1と並列処理用の複数のプロセッサ（CPU）2～4から成る。高速CPU1と3つの低速CPU2、3、4は各々、キャッシュメモリ5、6、7、8を介して共通バス9に接続されている。低速CPU2、3、4は同一機構（又は同一構成）より成るものとする。

【0014】このバス9には、共有メモリ10が接続されている。共有メモリ10には、処理プログラム11と並列動作度テーブル作成プログラム12及びプロセス割り付けプログラム13と並列動作度テーブル14が格納されている。また、プロセス実行頻度の情報として、プロセス実行頻度作成プログラム15とプロセス実行頻度テーブル16が格納されている。

5

【0015】ここで共通メモリ10内のプログラム11、12、13、15、及びテーブル14、16について簡単に説明する。詳細は追って説明する。処理プログラム11…マルチプロセッサシステムの処理対象とするプログラムのことであり、逐次型言語で作成されたプログラムを指す。このプログラム11は複数のプロセス $A_1 \sim A_n$ から成る。プロセス割り付け判定プログラム13…プロセスの並列動作度テーブル14（図5の14）を参照して、上記処理プログラムを構成するプロセス $A_1 \sim A_n$ について、マルチプロセッサシステムのCPU1～4への割り付けを行う。ここで割り付けとは、該当のCPUでプロセスを実行させることであり、プロセスのスケジューリング機能に相当する。並列動作度テーブル作成プログラム12…プロセスの並列動作度テーブル14（図5の14）を事前に作成するためのプログラムである。並列動作度テーブル14…並列動作度テーブル作成プログラム12によって作られた並列動作度を、プロセス対応にテーブル化して記憶したものである。プロセス実行頻度テーブル16…プロセス対応に、あるカウント時間内での実行回数を記録したテーブルである。このテーブル16は、処理プログラム10の実際の実行に先立って、得たものであり、並列動作度テーブル14の作成のために使う。具体例としての図3のテーブル16が該当する。プロセス実行頻度作成プログラム15…プロセス実行頻度テーブル16を作成するためのプログラムである。このプログラム15は、処理プログラム10の実際の実行に先立ち動作させる。

【0016】次に、図1のマルチプロセッサシステムの全体動作を説明する。

(1)、並列動作度テーブル14の作成…処理プログラム11の動作に先立って並列動作度テーブル14を作成する。この並列動作度テーブル14は、2段階で作成されるものであり、先ずプロセス実行頻度作成プログラム15によってプロセス実行頻度テーブル16を作成し、次いで並列動作度テーブル作成プログラム12によって、テーブル16を利用して並列動作度テーブル14を作成することになる。

(2)、処理プログラム11を構成するプロセスのマルチCPU1～4への割り付け…プロセス割り付け判定プログラム13が(1)で算出した並列動作度テーブル14を参照して、マルチCPU1～4へ、処理プログラム11を構成するプロセス $A_1 \sim A_n$ を割り付ける。割り付け終了後に、割り付けたプロセスを、割り付け先のキャッシュメモリ5～8に対応ずけて送り格納する。

(3)、処理プログラム11の実行…(2)の割り付け結果に従って、マルチCPU1～4の動作により逐次処理、並びに並列処理がなされ、プログラム11の実行をはかる。

【0017】並列動作度作成プログラム12の起動の時期は任意であり、オフライン的な処理で行う。また、図

6

1のマルチプロセッサシステムを利用して作成してもよく、図1とは異なる他のマルチプロセッサシステムによって作成しても良い。要は、図1のマルチプロセッサシステムとしての立ち上げ前に行えばよい。図2には、そうした一例を示し、図1のマルチプロセッサシステムを利用して並列動作度を自動作成する時の、並列動作度作成プログラム12の動作時期と動作内容とを示す計算機モード（マルチプロセッサシステムとしての計算機モードの意味）図である。ここで、図2の中でフロー24と25とが、並列動作度作成プログラム12である。

【0018】図2において、計算機が立ち上げ処理21を完了すると、フロー22に移り、並列動作度テーブル作成モードに入るか否かが表示画面に表示され、操作者はこれをみて、並列動作度テーブル作成モードに入るか否かの指示を行う。並列動作度テーブル作成モードにすると、並列動作度テーブル14の作成24を行い、次いでフロー25に移り、作成した並列動作度テーブル14の内容を磁気ディスク等の外部記憶装置に保存する。ここで作成した並列動作度テーブル14は、通常モード（フロー26）においてプロセスの並列度判定に用いられる。フロー22で、並列動作度テーブル作成モードにしなければ、外部記憶装置に保存されているすでに作成（フロー24、25によって）済みの並列動作度テーブル14を共有メモリ10に展開し、通常モードの処理26へ遷移する。通常モードでは、処理プログラム11による計算機の本来的業務処理を行う。以上の2つのモードの終了は計算機停止処理27により行う。並列度作成モードから通常モードへの切り替え（又はその逆）は、一旦、計算機を停止させる（フロー27）が、並列動作度作成モードはしばしば行うものではないので、この方式で十分である。

【0019】図3(a)は、プロセス実行頻度作成プログラム15の処理フロー、図3(b)はプロセス実行頻度テーブル16を示す図である。処理プログラム11のシミュレーションによる処理を行って各プロセス A_i ($i=1 \sim n$) についてプロセスが実行される毎にプロセスの実行回数をカウントし（フロー31）、プロセス実行頻度テーブル16の中味を更新する（フロー32）。プロセス実行頻度テーブル16は、プロセス毎の実行回数のカウント数 r とカウント期間 W （カウント開始時刻 t_s 及びカウント終了時刻 t_e との差分）から成る。カウント時間 W の期間中にそのプロセス A_i が何回実行されたかをカウント数 r で示している。

【0020】図4は、図2の並列動作度作成プログラム12の中の処理フロー24の詳細フローである。並列動作度テーブル作成モードでは、まず、フロー41によって、プロセス $A_1 \sim A_n$ における2つのプロセスのすべての組合せのそれぞれについて、2つの同一機構（又は同一機種。例えば図1のCPU2と3）のプロセッサで同時に実行できる度合いを示す並列処理可能度 λ_{ij} ($i =$

1~n, j=1~n, i≠j)を求める。このようにして任意の2つのプロセス並列処理可能度 λ_{ij} は、2つのプロセスを同時に2つのプロセッサでモニタランさせることにより求める。このようにして求めた並列処理可能度 λ_{ij} に対して図3(b)のプロセスの実行頻度を考慮することにより、各プロセスをプロセッサ1~4へ配分する際の指標となる並列動作度 λ_i を求める(フロー42、43)。図5(a)は並列処理可能度 λ_{ij} 、及び図5(b)は並列動作度 λ_i を示す図である。

【0021】次に、図5を用いて並列処理可能度 λ_{ij} (図5(a))及び並列動作度 λ_i (図5(b))について説明する。図5において、ユーザプログラム5のプロセスを A_1, A_2, \dots, A_n とする。プロセス A_i を単独に実行させた時の処理時間を t_i とする。またこれらの A_i, A_j ($1 \leq i, j \leq n$)からなるプロセスの組合せ52を考え、このプロセス A_i, A_j を2つのプロセッサで同時に動作させる。この時の A_i の処理時間を t_{ij} とする。プロセス A_i を単独に実行させた時の処理時間 t_i と、他のプロセス A_j と同時に実行させた時の処理時間 t_{ij} を比較した場合、その比 t_i/t_{ij} は1以下の値であり、この値が1に近いほど並列処理可能度が高いと考えられる。一般に他のプロセスと同時に実行したときは処理時間が長くなる($t_i \leq t_{ij}$)ためである。そこで、プロセス A_i のプロセス A_j に対する並列処理可能度 λ_{ij} を次の式により定義する。

【数1】 $\lambda_{ij} = t_i / t_{ij}$

こうして求めた並列処理可能度 λ_{ij} が図5(a)の内容である。

【0022】また、プロセス A_j の実行頻度(単位時間内での実行回数) μ_j は、プロセス実行頻度テーブル33の実行回数 r_j とカウント期間($t_s \sim t_e$)より、下記で算出する。

【数2】 $\mu_j = r_j / (t_s \sim t_e)$

次に、プロセス A_i をプロセッサへ配分する際の指標となる並列動作度 λ_i は、他のプロセスが動作している時にプロセス A_i が動作可能な度合いなので、上記、プロセス A_i のプロセス A_j に対する並列処理可能度 λ_{ij} とプロセス A_j の実行頻度 μ_j の関数として、例えば、次のように定義する。

【数3】

$$\lambda_i = \left(\sum_j (1/\mu_j) \cdot \lambda_{ij} \right) / \sum_j (1/\mu_j)$$

こうして求めた並列動作度 λ_i を処理時間 t_i と共に示したものが図5(b)である。ここで、実行頻度 μ_j が逆数(1回の実行時間となる)として扱われている理由は、プロセスの実行頻度が低いと他のプロセスが実行できる割合が高くなり、逆に、実行頻度が高いと他のプロセスが実行できる割合が低くなるからである。

【0023】図5(b)において、並列動作度 λ_i から以下の如き意味を持たせる。プロセス A_i が並列度大と

は、 λ_i がしきい値 ρ 以上となることを云う。プロセス A_i が並列度小とは、 λ_i がしきい値 ρ 以下となることを云う。 ρ の値は、プロセスの並列動作度 λ_i とプロセッサの台数 m から以下のようにして決める。 $\{\lambda_1, \dots, \lambda_n\}$ を λ_i の降順(小さい値順)に並べたものを $\lambda_{i'}$
 $\geq \dots \geq \lambda_{n'}$ とする時、この値が上位 $(1/m) \cdot 100$ (%)に入る $\lambda_{i'}$ が高速プロセッサで処理する対象となるので、上位 $(1/m) \cdot 100$ (%)に入る $\lambda_{i'}$ の最小の値を ρ とする。本実施の形態では、 $m=4$ であるから ρ は上位25%に入る $\lambda_{i'}$ の最小値となる。

【0024】図6は、プロセス割り付け判定プログラム13の処理フローである。このプロセス割り付けとはプログラムの実際の処理を行う過程での動的な割り付けを指す。プロセスが実行される時、プロセス割り付け判定プログラム13は、実行するプロセスをキュー(図6フロー601。いわゆる待行列のこと)より取得する。例えばキューの先頭にあるプロセスを取得する。この時、当該プロセスの並列動作度テーブル14の情報(動作度の大小)を参照して(フロー603)並列度が小であれば高速プロセッサが空くのを待って、プロセスを高速プロセッサに割り付ける(フロー604)。並列度が大きければ、プロセッサ数分の並列度大のプロセス(本実施の形態では、4個)をキュー(待行列)より取り出し(フロー605~フロー611)、並列動作度テーブル14の処理時間を参照して、実行させる並列度大のプロセスを処理時間順に並べ、全プロセッサが空くのを待って、処理時間が最長のプロセスを高速プロセッサへ、残りを高速でないプロセッサへ割り付ける(フロー612)。

【0025】図7は、ある処理プログラム11(メインプログラム)の概略構成図である。この図は、図6に基づき、逐次形と並列形とに割り付けた結果を示す図である。メインプログラム11は、逐次処理プログラム S_1 と、この逐次処理プログラム終了後に実行される並列処理プログラム $P_1 \sim P_4$ と、この並列処理プログラム終了後に実行される逐次処理プログラム S_2 で構成されている。各プログラム $S_1, P_1 \sim P_4, S_2$ は、前述のプロセス A_i ($i=1 \sim 6$)に相当する。ここでプロセスとは、プログラムの実行単位のこと、プログラムを実行するのに必要な情報(CPUが機械語の命令列として解釈できるテキストとデータ等)から成る。

【0026】各プログラム $S_1, S_2, P_1 \sim P_4$ の並列処理可能度 λ_{ij} は、プログラム S_1, S_2 が逐次プログラム、プログラム $P_1 \sim P_4$ が並列プログラムとした場合、以下になる(又は、なるものとする)。尚、 λ_{ij} の代わりに、印字の関係上 $\lambda(S, P)$ や $\lambda(P, P)$ 、 $\lambda(P, S)$ 、 $\lambda(S, S)$ で示すことにする。当然にカッコ内の手前の英字が i 、後ろの英字が j を示す。

$\lambda(S_1, P_1) = \lambda(S_1, P_2) = \lambda(S_1, P_3) = \lambda$

9

$$(S_1, P_4) = \lambda(S_1, S_2) = 0$$

$$\lambda(P_1, S_1) = \lambda(P_1, S_2) = 0$$

$$\lambda(P_1, P_2) = \lambda(P_1, P_3) = \lambda(P_1, P_4) = 1$$

$$\lambda(P_3, S_1) = \lambda(P_2, S_2) = 0$$

$$\lambda(P_2, P_1) = \lambda(P_2, P_3) = \lambda(P_2, P_4) = 1$$

$$\lambda(P_3, S_1) = \lambda(P_3, S_2) = 0$$

$$\lambda(P_3, P_1) = \lambda(P_3, P_2) = \lambda(P_3, P_4) = 1$$

$$\lambda(P_4, S_1) = \lambda(P_4, S_2) = 0$$

$$\lambda(P_4, P_1) = \lambda(P_4, P_2) = \lambda(P_4, P_3) = 1$$

また、各プログラムの実行頻度 μ は、下記の如くすべてのプロセッサで全て等しいとする。

$$\mu(S_1) = \mu(P_1) = \mu(P_2) = \mu(P_3) = \mu(P_4) = \mu(S_2) = \mu_0$$

このとき、各プログラムの並列動作度は、各々以下のようになる。

$$\lambda(S_1) = \lambda(S_2) = 0$$

$$\lambda(P_1) = \lambda(P_2) = \lambda(P_3) = \lambda(P_4) = 4$$

この場合のしきい値は、 $\rho = 4$ となる。従って、このプログラム11は、並列動作度作成プログラムにより並列度小($\lambda_i < \rho$)の逐次処理プログラム S_1 、 S_2 と並列度大($\lambda_i \geq \rho$)の並列処理プログラム $P_1 \sim P_4$ に分類される。

【0027】図8は、図1に示した4個のCPU1~4(CPU1は高速)でメインプログラム11を実行する時の、各プロセスCPUへの配分を説明する図である。今、並列処理プログラム $P_1 \sim P_4$ の処理時間 $t_1 \sim t_4$ を $t_3 \leq t_1 \leq t_2 \leq t_4$ とする。プロセス割り付け処理は、プロセスの並列度の大小と処理時間の長短により行うので高速プロセッサCPU1には、逐次処理プログラム S_1 、 S_2 及び処理時間の長い並列処理プログラム P_4 、プロセッサCPU2~CPU4には、処理時間の短い並列処理プログラム $P_1 \sim P_3$ が割り付けられる。これにより、逐次処理と並列処理が混在するプログラムをそれを処理するのに適切なプロセッサに配分でき、全体として処理性を上げることができる。

【0028】具体的な処理例としては、電力系統の計画支援業務がある。電力系統に送電線停止等の事故が起きたことを想定した場合、その復旧手順(系統のどのルートから回復したら良いか等)は複数の手段があり、計算機ではそれに対応して複数の復旧方針作成プログラム(例：系統の切り替えによる復旧、発電機の調整による復旧等)がある。このプログラムは前処理から計算に必要なデータをもらおうと並列に処理できるので、

・逐次処理プログラム1(S_1)…前処理(事故の特定、復旧範囲の決定)

・並列処理プログラム1(P_1)…復旧方針作成プログラム1

・並列処理プログラム2(P_2)…復旧方針作成プログラム2

・並列処理プログラム3(P_3)…復旧方針作成プログ

10

ラム3

・並列処理プログラム4(P_4)…復旧方針作成プログラム4

・逐次処理プログラム2(S_2)…後処理(結果の比較、評価)

となる。4つの復旧方針作成プログラムの中では、処理の複雑なものが処理時間がかかるので、この処理が高速プロセッサに割り当てられる。

【0029】図9は、メインプログラム11を従来の構成のCPU(4個の同じ性能のCPU)で従来手法のプロセス配分を行った場合を示す。従来手法は、CPUの性能が同じなので逐次処理 S_1 及び S_2 がCPU1で動作している間、他のCPU2~CPU4の空き時間が逐次用の高速プロセッサを使用する構成に比べて大きい。また、並列処理プログラム $P_1 \sim P_4$ の実行においてもCPU性能が同じため、並列処理時間のばらつきを考慮していないので、CPU資源の無駄が生じる。この現象は、逐次処理の処理時間が長いほど、又、並列処理時間のばらつきが大きいほど顕著になる。

【0030】尚、上記実施の形態では、高速1台、低速複数台としたが、高速2台以上の例での配分例へも拡張できる。また、低速複数台は同一機構又は同一構成としたが、そうでない例もあり、そうした能力に応じた割り付けを行ってもよい。また、逐次を高速、並列を低速に割り当てるものとしたが、低速と高速との区分をせずに、逐次を1台、並列を残りのものに割り当てる如きやり方もある。

【0031】

【発明の効果】本発明によれば、従来の逐次型言語のプログラムをそのまま使用してプロセス間の並列処理を容易に実現し、逐次処理と並列処理が混在するプログラムを高速プロセッサと並列処理用の複数プロセッサの2種類のハード資源に有効に配分でき、トータルとして効率的なマルチプロセッサシステムができるという効果がある。

【図面の簡単な説明】

【図1】本発明のマルチプロセッサシステムの構成例図である。

【図2】並列動作度作成プログラムの動作を含む計算モードのフローチャート図である。

【図3】プロセス実行頻度作成プログラムでの処理フロー及びプロセス実行頻度テーブルを示す図である。

【図4】並列動作度テーブル作成プログラムの処理フロー例図である。

【図5】並列処理可能度テーブル、並列動作度テーブルのデータ例図である。

【図6】プロセス割り付け処理フロー図である。

【図7】プログラム割り付け例図である。

【図8】CPUへのプログラム割り付け例図である。

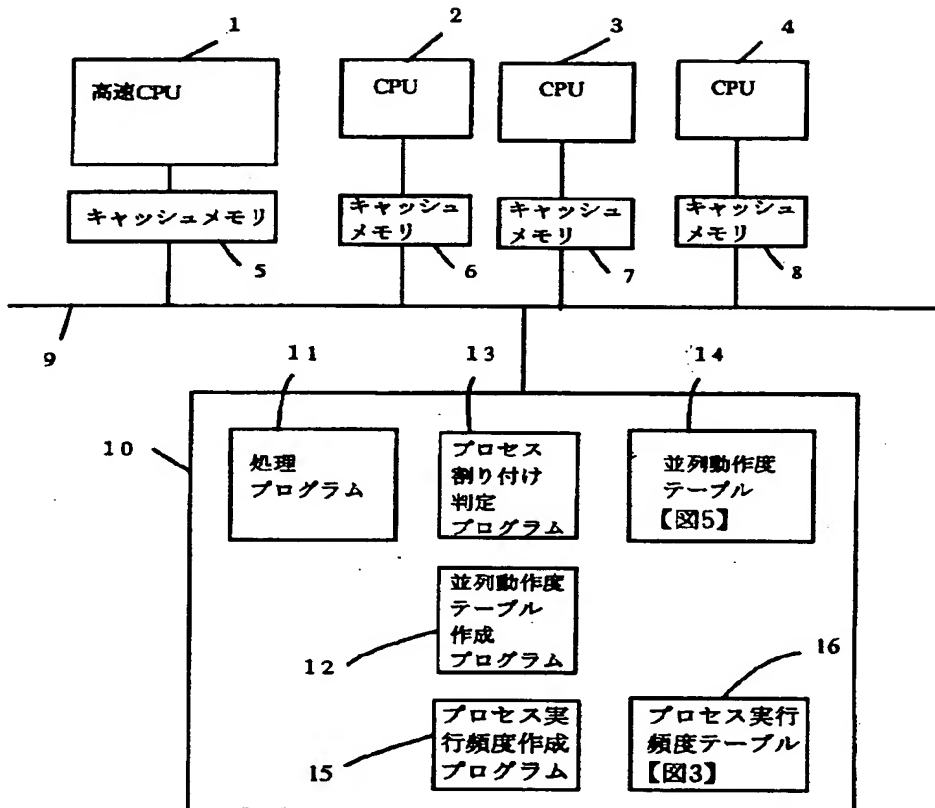
【図9】従来例によるプログラム割り付け例図である。

【符号の説明】

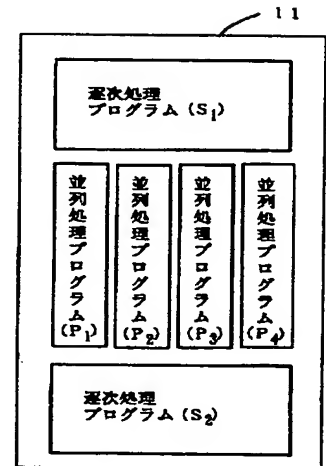
- 1 高速CPU
2、3、4 低速CPU
5、6、7、8 キャッシュメモリ
9 共通バス
10 共有メモリ

- * 11 処理プログラム
12 並列動作度テーブル作成プログラム
13 プロセス割り付け判定プログラム
14 並列動作度
15 プロセス実行頻度作成プログラム
* 16 プロセス実行頻度テーブル

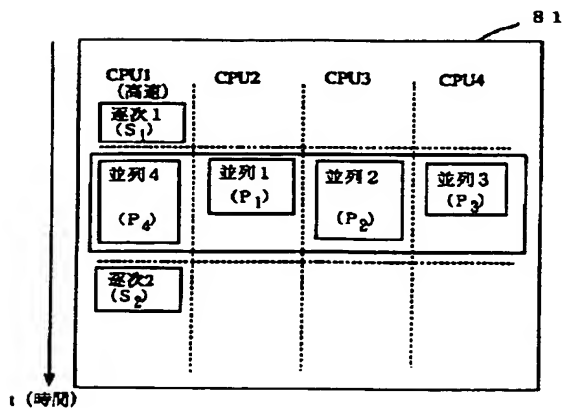
【図1】



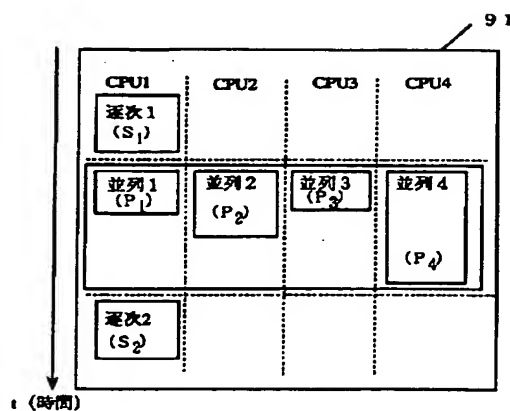
【図7】



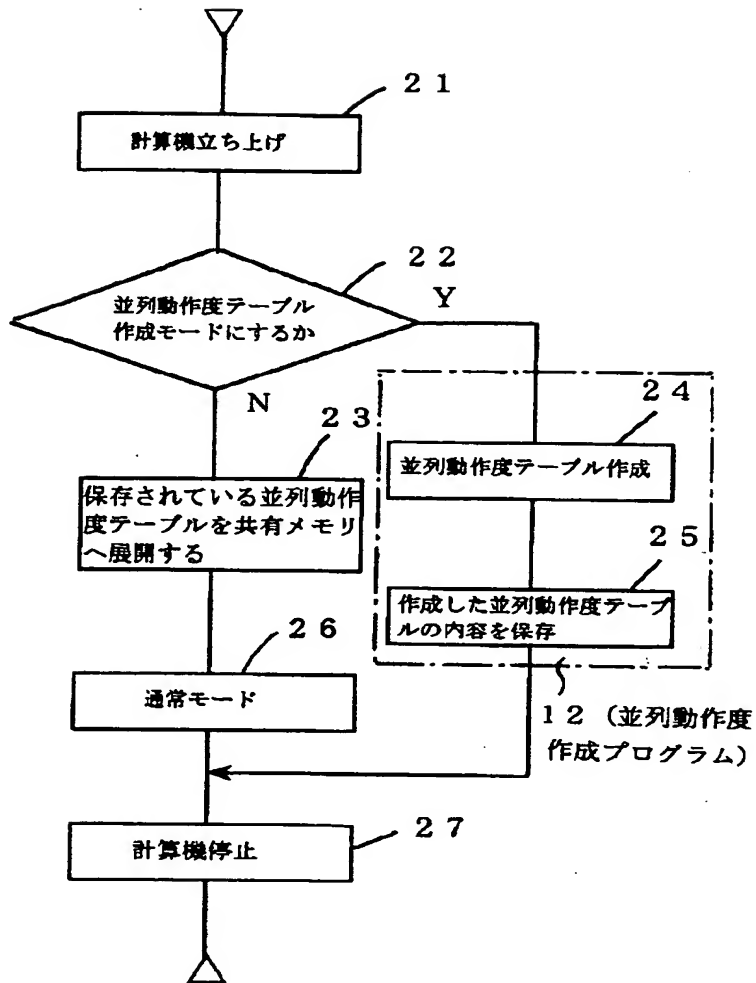
【図8】



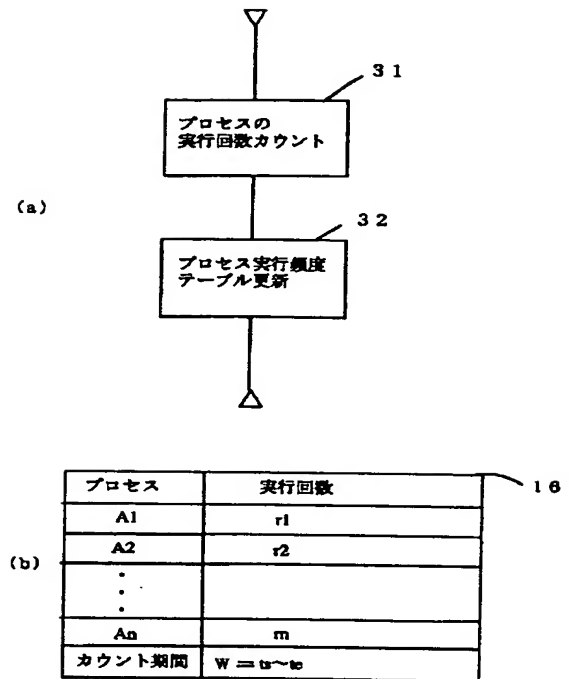
【図9】



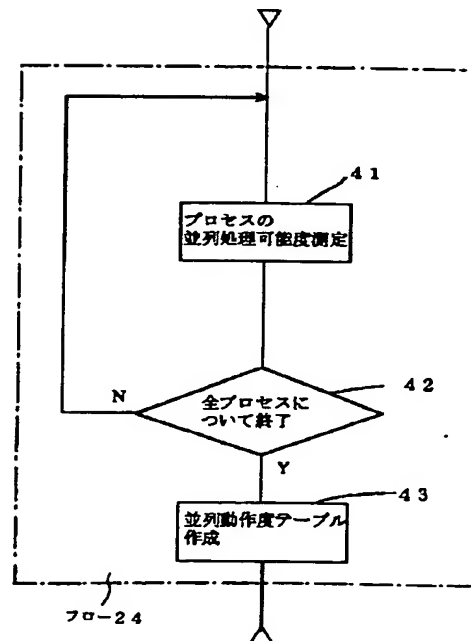
【図2】



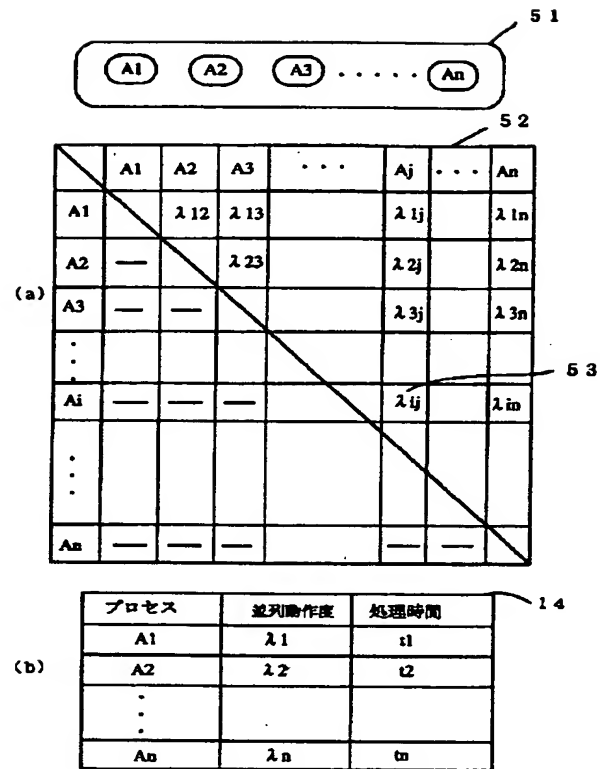
【図3】



【図4】



【図5】



【図6】

